

# 1 Compiler

The Compiler doesn't really do much compiling, but rather just strips off the proof code after ensuring that it is valid.

---

```
module Compiler where
  import Data.List
  import Data.List.Split
  import Parser
  import Analyzer
  import Lexer (lexify)
```

---

Lines of proof code are identified by a prefix, ">" by default

---

```
lineStart :: String
lineStart = ">"
```

---

The first step of “compiling” is then to separate the type code from the actual code, based on the prefix. They don't actually need to be kept together, as they are completely independent systems and will be linked together purely on a named basis. So long as the actual code and proof code come in corresponding order, all can be resolved without much trouble.

---

```
takeCode :: String → String → String
takeCode proofLine file =
  intercalate "\n" (filter (not ∘ isPrefixOf proofLine) (splitOn "\n"
    file))

takeProofs :: String → String → String
takeProofs proofLine file =
  intercalate "\n" $
    map (drop $ length proofLine) $
    filter (isPrefixOf proofLine) $
    splitOn "\n" file
```

---

Having separated the actual code from the proof code, it is then passed to the Parser, and then the Analyzer to ensure that the proofs check out, before the actual code is returned, with all proofs stripped out.

---

```
check :: AST → Either CompileError Bool
check ast =
  case analyze ast of
    Right True      → Right True
    Left (AnalysisError message) → Left $ CompileError $ "Could not
    compile. " ++ message

compile :: String → Either CompileError String
compile file = case check (parseProofs proof 'annotates' parseCode code
) of
  Right True → Right code
  Left err   → Left err
  where (proof, code) = (takeProofs lineStart file, takeCode lineStart
    file)
```

---

If analysis fails, a `CompileError` is returned instead, allowing for the programmer to be notified of what went wrong.

---

```
-- TODO: investigate proper error handling methods
newtype CompileError = CompileError String
```

---